Deep Learning: Homework 3

Author: Rudy Martinez

Date: 10/26/2021

Google Colab Link: https://colab.research.google.com/drive/1l50xZSOZ0wi1IHvLn2EDbTnHs8bFtgV2?usp=sharing

Libraries

```
#Data Manipulation
import pandas as pd
import numpy as np

#Data Viz
import matplotlib.pyplot as plt

#Machine Learning
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.datasets import imdb
from tensorflow.keras import models
from tensorflow.keras import layers
from tensorflow.keras import optimizers
from tensorflow.keras import losses
from tensorflow.keras import metrics
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler
```

P1 (40pt): In the code example of "Classifying movie reviews" explained in Lecture 6, make the following changes sequentially to the two neural network models in the example:

0. **Setup - Load `imbd` Data**

```
#The following code will load the dataset
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words=10000)
```

```
def vectorize_sequences(sequences, dimension=10000):
    # Create an all-zero matrix of shape (len(sequences), dimension)
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1.  # set specific indices of results[i] to 1s
    return results

# Our vectorized training data and testing data
x_train = vectorize_sequences(train_data)
x_test = vectorize_sequences(test_data)

# Our vectorized labels
y_train = np.asarray(train_labels).astype('float32')
y_test = np.asarray(test_labels).astype('float32')
```

1. **Change the number of neurons on the two hidden layers to 32 units. (10pt)**
2. **Use the tanh activation (an activation that was popular in the early days of neural networks) instead of relu for the two hidden layers. (10pt)**
3. **Add an additional hidden layer with 32 units and tanh activation function. (10pt)**

```
#Keras Implementation
model = models.Sequential()
model.add(layers.Dense(32, activation='tanh', input_shape=(10000,)))
model.add(layers.Dense(32, activation='tanh'))                        #Change to tanh
model.add(layers.Dense(32, activation='tanh'))                        #Add additional layer
model.add(layers.Dense(1, activation='sigmoid'))
```

4. **Retrain the newly defined models and evaluate the trained models on the testing dataset to get the accuracy. (10pt)**

```
#Configure model with the `rmsprop` optimizer and the `binary_crossentropy` loss function.
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])

#Pass optimizer, loss function and metrics as strings
model.compile(optimizer=optimizers.RMSprop(learning_rate=0.001),
              loss='binary_crossentropy',
              metrics=['accuracy'])

model.compile(optimizer=optimizers.RMSprop(learning_rate=0.001),
              loss=losses.binary_crossentropy,
              metrics=[metrics.binary_accuracy])
```

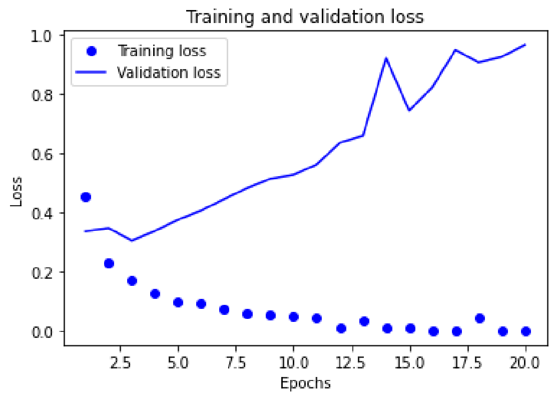
```
#Create a "validation set" by setting apart 10,000 samples from the original training data
x_val = x_train[:10000]
partial_x_train = x_train[10000:]
```

```
y_val = y_train[:10000]
partial_y_train = y_train[10000:]

#Train Model
history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))

#History Dictionary
history_dict = history.history
```

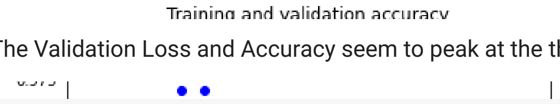
```
Epoch 1/20
30/30 [==============================] - 3s 75ms/step - loss: 0.4530 - binary_accuracy: 0.8010 - val_loss: 0.3354 - val_binary_accuracy: 0.8570
Epoch 2/20
30/30 [==============================] - 2s 53ms/step - loss: 0.2308 - binary_accuracy: 0.9107 - val_loss: 0.3457 - val_binary_accuracy: 0.8616
Epoch 3/20
30/30 [==============================] - 2s 53ms/step - loss: 0.1694 - binary_accuracy: 0.9382 - val_loss: 0.3037 - val_binary_accuracy: 0.8826
Epoch 4/20
30/30 [==============================] - 1s 44ms/step - loss: 0.1241 - binary_accuracy: 0.9573 - val_loss: 0.3363 - val_binary_accuracy: 0.8809
Epoch 5/20
30/30 [==============================] - 1s 42ms/step - loss: 0.0992 - binary_accuracy: 0.9673 - val_loss: 0.3741 - val_binary_accuracy: 0.8760
Epoch 6/20
30/30 [==============================] - 1s 43ms/step - loss: 0.0938 - binary_accuracy: 0.9666 - val_loss: 0.4051 - val_binary_accuracy: 0.8685
Epoch 7/20
30/30 [==============================] - 1s 46ms/step - loss: 0.0714 - binary_accuracy: 0.9776 - val_loss: 0.4437 - val_binary_accuracy: 0.8725
Epoch 8/20
30/30 [==============================] - 2s 53ms/step - loss: 0.0578 - binary_accuracy: 0.9822 - val_loss: 0.4815 - val_binary_accuracy: 0.8696
Epoch 9/20
30/30 [==============================] - 1s 45ms/step - loss: 0.0543 - binary_accuracy: 0.9838 - val_loss: 0.5122 - val_binary_accuracy: 0.8698
Epoch 10/20
30/30 [==============================] - 1s 49ms/step - loss: 0.0507 - binary_accuracy: 0.9849 - val_loss: 0.5261 - val_binary_accuracy: 0.8650
Epoch 11/20
30/30 [==============================] - 1s 46ms/step - loss: 0.0444 - binary_accuracy: 0.9882 - val_loss: 0.5605 - val_binary_accuracy: 0.8674
Epoch 12/20
30/30 [==============================] - 1s 49ms/step - loss: 0.0081 - binary_accuracy: 0.9990 - val_loss: 0.6337 - val_binary_accuracy: 0.8649
Epoch 13/20
30/30 [==============================] - 1s 43ms/step - loss: 0.0325 - binary_accuracy: 0.9911 - val_loss: 0.6578 - val_binary_accuracy: 0.8609
Epoch 14/20
30/30 [==============================] - 1s 42ms/step - loss: 0.0095 - binary_accuracy: 0.9980 - val_loss: 0.9192 - val_binary_accuracy: 0.8325
Epoch 15/20
30/30 [==============================] - 1s 42ms/step - loss: 0.0077 - binary_accuracy: 0.9975 - val_loss: 0.7424 - val_binary_accuracy: 0.8604
Epoch 16/20
30/30 [==============================] - 1s 43ms/step - loss: 9.1121e-04 - binary_accuracy: 1.0000 - val_loss: 0.8208 - val_binary_accuracy: 0.8591
Epoch 17/20
30/30 [==============================] - 1s 42ms/step - loss: 3.5325e-04 - binary_accuracy: 1.0000 - val_loss: 0.9472 - val_binary_accuracy: 0.8549
Epoch 18/20
30/30 [==============================] - 1s 50ms/step - loss: 0.0437 - binary_accuracy: 0.9917 - val_loss: 0.9051 - val_binary_accuracy: 0.8597
Epoch 19/20
30/30 [==============================] - 1s 43ms/step - loss: 2.1778e-04 - binary_accuracy: 1.0000 - val_loss: 0.9238 - val_binary_accuracy: 0.8592
Epoch 20/20
30/30 [==============================] - 1s 43ms/step - loss: 1.4957e-04 - binary_accuracy: 1.0000 - val_loss: 0.9635 - val_binary_accuracy: 0.8578
```

```
#Training and Validation Loss
acc = history.history['binary_accuracy']
val_acc = history.history['val_binary_accuracy']
```

```
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



```
#Training and Validation Accuracy
plt.clf()   # clear figure
acc_values = history_dict['binary_accuracy']
val_acc_values = history_dict['val_binary_accuracy']
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



- The Validation Loss and Accuracy seem to peak at the third epoch. To avoid "overfitting", we will change epochs to 3.

```
#Change Epochs to 3 and View Results
model = models.Sequential()
model.add(layers.Dense(32, activation='tanh', input_shape=(10000,)))
model.add(layers.Dense(32, activation='tanh'))
model.add(layers.Dense(32, activation='tanh'))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=3, batch_size=512, verbose = 0)
results = model.evaluate(x_test, y_test)
results
```

```
782/782 [==============================] - 2s 2ms/step - loss: 0.3457 - accuracy: 0.8636
[0.3456971049308777, 0.8636000156402588]
```

- This revised approach achieved an **accuracy of 86.36%**

```
#Generate the likelihood of reviews being positive by using the `predict` method
model.predict(x_test)
```

```
array([[0.08507851],
       [0.9959099 ],
       [0.1650086 ],
       ...,
       [0.09225032],
       [0.02167743],
       [0.35031825]], dtype=float32)
```

P2 (60pt): Write a Python code in Colab using NumPy, Panda, Scikit-Learn and Keras to complete the following tasks:

1. Import the Auto MPG dataset using pandas.read_csv(), use the attribute names as explained in the dataset description as the column names, view the strings ‘?’ as the missing value, and whitespace (i.e., ‘\s+’) as the column delimiter. Print out the shape and first 5 rows of the DataFrame. (5pt)

```
#Set Random Seed
np.random.seed(123)

#Read Data from URL (Implement Pre-defined Headings)
```

```
mpg_data = pd.read_csv('http://archive.ics.uci.edu/ml/machine-learning-databases/auto-mpg/auto-mpg.data',
                       sep='\s+', header=None,
                       names=['mpg','cylinders','displacement','horsepower','weight','acceleration','model_year','origin','car_name'],
                       na_values = ['?'])

#Create Dataframe
mpg_df = mpg_data.copy()

print("Data Shape: ",mpg_df.shape, "\n\n")
mpg_df.head()
```

```
Data Shape:  (398, 9)
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	car_name
0	18.0	8	307.0	130.0	3504.0	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165.0	3693.0	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150.0	3436.0	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150.0	3433.0	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140.0	3449.0	10.5	70	1	ford torino

2. Delete the “car_name” column using .drop() and drop the rows containing NULL value using .dropna(). Print out the shape of the DataFrame. (5pt)

```
#Drop `car_name`
mpg_df.drop(columns=['car_name'], axis = 1, inplace=True)
```

```
#Drop Null Values
mpg_df = mpg_df.dropna()
```

```
#Check for Null Values
print(mpg_df.isnull().sum(),"\n\n")

#Check Shape
print("Data Shape: ",mpg_df.shape)
```

```
mpg             0
cylinders       0
displacement    0
horsepower      0
weight          0
acceleration    0
model_year      0
origin          0
dtype: int64
```

```
Data Shape:  (392, 8)
```

3. For the ‘origin’ column with categorical attribute, replace it with the columns with numerical attributes using one-hot encoding. Print out the shape and first 5 rows of the new DataFrame. (5pt)

```
#OneHotEncoding - Initialize Encoder
encoder = OneHotEncoder(sparse=False)

#Specify Categorical Column
columns = ['origin']

#Apply Encoder
df_encoded = pd.DataFrame(encoder.fit_transform(mpg_df[columns]))
df_encoded.columns = encoder.get_feature_names(columns)

#Remove `origin` Column
mpg_df.drop(["origin"] ,axis=1, inplace=True)

#Reset and Drop Index
mpg_df.reset_index(drop=True, inplace=True)

#Concatenate the OneHotEncoded Columns
mpg_df = pd.concat([mpg_df, df_encoded], axis=1)

#Print Shape and First Five Rows
print("Data Shape: ",mpg_df.shape, "\n\n")
mpg_df.head()
```

```
Data Shape:  (392, 10)
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin_1	origin_2	origin_3
0	18.0	8	307.0	130.0	3504.0	12.0	70	1.0	0.0	0.0
1	15.0	8	350.0	165.0	3693.0	11.5	70	1.0	0.0	0.0
2	18.0	8	318.0	150.0	3436.0	11.0	70	1.0	0.0	0.0
3	16.0	8	304.0	150.0	3433.0	12.0	70	1.0	0.0	0.0
4	17.0	8	302.0	140.0	3449.0	10.5	70	1.0	0.0	0.0

4. Separate the “mpg” column from other columns and view it as the label vector and others as the feature matrix. Split the data into a training set (80%) and testing set (20%) using train_test_split and print out their shapes. Print out the statistics of your training feature matrix using .describe(). (5pt)

```
#Create Label Vector and Feature Matrix
label_vector = mpg_df['mpg']
feature_matrix = mpg_df.drop("mpg", axis=1)

#Split Data Using train_test_split
```

```
#Split Data Using train_test_split
X_train, X_test, y_train, y_test = train_test_split(feature_matrix, label_vector, test_size=0.2, random_state=42)

#Print out Shapes
print("X_train: ", X_train.shape, " | ", "y_train: ", y_train.shape, " | ", "X_test: ", X_test.shape, " | ", "y_test: ", y_test.shape, "\n\n")

#Print Summary Statistics
X_train.describe()
```

```
X_train:  (313, 9)  |  y_train:  (313,)  |  X_test:  (79, 9)  |  y_test:  (79,)
```

	cylinders	displacement	horsepower	weight	acceleration	model_year	origin_1	origin_2	origi
count	313.000000	313.000000	313.000000	313.000000	313.000000	313.000000	313.000000	313.000000	313.000
mean	5.482428	195.517572	104.594249	2986.124601	15.544089	76.207668	0.645367	0.153355	0.201
std	1.700446	103.766567	38.283669	841.133957	2.817864	3.630136	0.479168	0.360906	0.401
min	3.000000	70.000000	46.000000	1613.000000	8.000000	70.000000	0.000000	0.000000	0.000
25%	4.000000	105.000000	76.000000	2234.000000	13.500000	73.000000	0.000000	0.000000	0.000
50%	4.000000	151.000000	95.000000	2855.000000	15.500000	76.000000	1.000000	0.000000	0.000
75%	8.000000	302.000000	129.000000	3645.000000	17.300000	79.000000	1.000000	0.000000	0.000
max	8.000000	455.000000	230.000000	5140.000000	24.800000	82.000000	1.000000	1.000000	1.000

5. Normalize the feature columns in both training and testing datasets so that their means equal to zero and variances equal to one. Note that the testing set can only be scaled by the mean and standard deviation values obtained from the training set. Describe the statistics of your normalized feature matrix of training dataset using .describe() in Pandas. (5pt)
   - Option 1: You can follow the normalization steps in the code example of “Predicting house prices: a regression example” in Lecture 6.
   - Option 2: You can use StandardScaler() in Scikit-Learn as in Homework 2 but you may need to transform a NumPy array back to Pandas DataFrame using pd.DataFrame() before calling .describe().

```
#Instantiate Scaler
scaler = StandardScaler()

#Transform X_train and X_test
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

#Summary Statistics for Normalized Feature Matrix
feature_stats = pd.DataFrame(X_train, columns=feature_matrix.columns)
feature_stats.describe()
```

	cylinders	displacement	horsepower	weight	acceleration	model_year	origin_1	origi
count	3.130000e+02	3.130000e+02	3.130000e+02	3.130000e+02	3.130000e+02	3.130000e+02	3.130000e+02	3.130000e
mean	-4.469268e-17	3.830801e-17	-1.191805e-16	2.468739e-16	5.242523e-16	-1.787353e-15	1.624544e-16	-7.62613
std	1.001601e+00	1.001601e+00	1.001601e+00	1.001601e+00	1.001601e+00	1.001601e+00	1.001601e+00	1.001601e
min	-1.462206e+00	-1.211552e+00	-1.532979e+00	-1.635082e+00	-2.681524e+00	-1.712775e+00	-1.349007e+00	-4.25596
25%	-8.731837e-01	-8.737161e-01	-7.481006e-01	-8.956112e-01	-7.265654e-01	-8.850368e-01	-1.349007e+00	-4.25596

6. Build a sequential neural neural network model in Keras with two densely connected hidden layers (32 neurons and ReLU activation function for each hidden layer), and an output layer that returns a single, continuous value. Print out the model summary using .summary(). (10pt)

```
#Build Model
model = models.Sequential()
model.add(layers.Dense(32, activation='relu', input_shape=(X_train.shape[1],)))
model.add(layers.Dense(32, activation='relu'))
model.add(layers.Dense(1))
model.summary()
```

```
Model: "sequential_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_26 (Dense)             (None, 32)                320
_________________________________________________________________
dense_27 (Dense)             (None, 32)                1056
_________________________________________________________________
dense_28 (Dense)             (None, 1)                 33
=================================================================
Total params: 1,409
Trainable params: 1,409
Non-trainable params: 0
_________________________________________________________________
```

7. Define the appropriate loss function, optimizer, and metrics for this specific problem and compile the NN model. (10pt)

```
model.compile(optimizer='rmsprop', loss='mse', metrics=['mae'])
```

- Note that we are compiling the network with the `mse` loss function -- Mean Squared Error, the square of the difference between the predictions and the targets, a widely used loss function for regression problems.
- We are also monitoring a new metric during training: `mae`. This stands for Mean Absolute Error. It is simply the absolute value of the difference between the predictions and the targets.

8. Put aside 20% of the normalized training data as the validation dataset by setting validation_split = 0.2 and set verbose = 0 to compress the model training status in Keras .fit(). Train the NN model for 100 epochs and batch size of 32 and plot the training and validation loss progress with respect to the epoch number. (10pt)

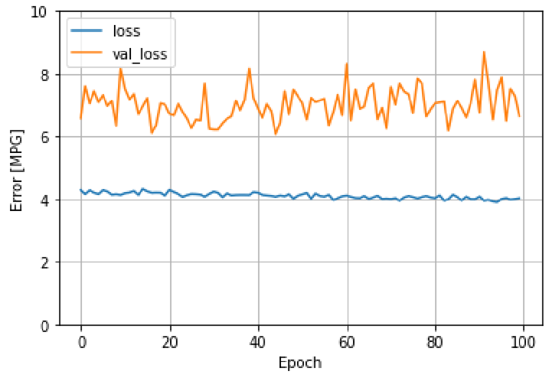
```
history = model.fit(X_train, y_train, epochs=100, batch_size=32, validation_split=0.20, verbose=0)
results = model.evaluate(X_test, y_test)
results
```

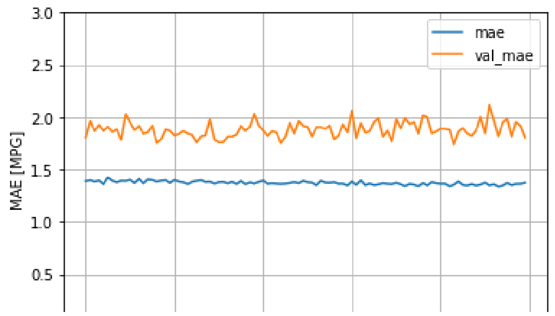
```
3/3 [==============================] - 0s 3ms/step - loss: 6.0362 - mae: 1.7510
[6.036228656768799, 1.7509526014328003]
```

```
hist = pd.DataFrame(history.history)
hist['epoch'] = history.epoch
hist.tail()
```

	loss	mae	val_loss	val_mae	epoch
95	4.002573	1.371548	7.878791	1.983825	95
96	4.031318	1.351101	6.494071	1.816319	96
97	3.981512	1.362548	7.520033	1.953430	97
98	3.999630	1.363418	7.279851	1.915742	98
99	4.026089	1.373653	6.642659	1.806405	99

```
def plot_loss(history):
  plt.figure()
  plt.plot(history.history['loss'], label='loss')
  plt.plot(history.history['val_loss'], label='val_loss')
  plt.ylim([0, 10])
  plt.xlabel('Epoch')
  plt.ylabel('Error [MPG]')
  plt.legend()
  plt.grid(True)

  plt.figure()
  plt.plot(history.history['mae'], label='mae')
  plt.plot(history.history['val_mae'], label='val_mae')
  plt.ylim([0, 3])
  plt.xlabel('Epoch')
  plt.ylabel('MAE [MPG]')
  plt.legend()
  plt.grid(True)

plot_loss(history)
```

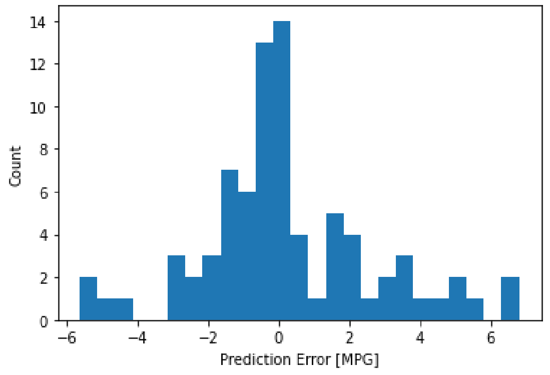




9. Use the trained NN model to make predictions on the normalized testing dataset and observe the prediction error. (5pt)

```
test_predictions = model.predict(X_test).flatten()

#Results Chart
plt.scatter(y_test, test_predictions)
plt.xlabel('True Values [MPG]')
plt.ylabel('Predictions [MPG]')
plt.axis('equal')
plt.axis('square')
plt.xlim([0,plt.xlim()[1]])
plt.ylim([0,plt.ylim()[1]])
_ = plt.plot([-100, 100], [-100, 100])
```

```
#Error
error = test_predictions - y_test
plt.hist(error, bins = 25)
plt.xlabel("Prediction Error [MPG]")
_ = plt.ylabel("Count")
```


Count
Prediction Error [MPG]